

- Pour un examen, le **plagiat** inclut le fait de demander de l'aide à quelqu'un pour répondre aux questions et le copier-coller venant de sources dont vous ne possédez pas tous les droits (i.e. TP fait en équipe, notes de cours, site Internet), sans en citer la source. Toute personne qui aide un autre étudiant pendant l'examen commet une **fraude**.
- En cas de doute sur le sens d'une question, énoncez clairement toutes suppositions que vous faites, soit en commentaires dans le code, soit dans la dernière question de cet examen (qui est un espace pour écrire vos commentaires). Nous ne répondrons pas aux questions.
- En cas de réel problème, vous pouvez écrire en message direct à "François-R Boyer" sur le Slack du cours.
- Vous avez droit à toutes les notes de cours et, pour les questions de «programmation» (où vous devez écrire un programme) à un compilateur, mais CodeRunner est suffisant pour répondre et les questions n'ont pas été particulièrement faites pour l'utilisation d'un autre compilateur. Vos programmes doivent passer les tests CodeRunner.
- La sauvegarde se fait automatiquement lorsque vous changez de page, elle est déclenchée quand le bouton « Suivant » dans le bas de la page est utilisé ou lorsque vous changez de page en utilisant le bloc «Navigation du test ». Cette sauvegarde permet de limiter la perte d'information en cas de coupure avec Internet ou autre problème technique et de garder la session active. Après deux heures d'inactivité (c.-à-d. aucune interaction avec le serveur), vous êtes automatiquement déconnecté(e) de Moodle.
- En cas de perte de connexion à la fin de l'examen, la tentative est envoyée automatiquement.
- L'examen est sur 20 points, le barème de chaque exercice est indiqué dans le bloc «Navigation du test ».

Bon travail !

Indiquez l'ordre de complexité de temps des algorithmes ci-dessous. On considère l'ordre moyen et/ou amorti. S'il y a plusieurs possibilités, indiquez l'ordre qui est le plus petit possible.

La fonction f a un temps en $O(1)$ et *range* n'ajoute pas de complexité par rapport à une boucle faite à la main.

Question **1**

Correct

Note de 1,00
sur 1,00

```
unordered_map<int,int> v;  
// ... des valeurs dans v
```

```
int n = v.size();  
for (int i : range(42))  
    v[f(i)] = i;
```

Est $O($ $)$ 

Votre réponse est correcte.

La réponse correcte est :

```
unordered_map<int,int> v;  
// ... des valeurs dans v
```

```
int n = v.size();  
for (int i : range(42))  
    v[f(i)] = i;
```

Est $O([1])$

Question **2**

Correct

Note de 1,00
sur 1,00

```
int somme = 0;  
for (int i : range(n))  
    for (int j : range(i))  
        somme += f(i,j);
```

Est $O($ $)$ 

Votre réponse est correcte.

La réponse correcte est :

```
int somme = 0;  
for (int i : range(n))  
    for (int j : range(i))  
        somme += f(i,j);
```

Est $O([n^2])$



Question **3**

Correct

Note de 1,00
sur 1,00

```
vector<int> v;  
int cnt = 0;  
for (auto&& e1 : v) {  
    cnt += count_if(  
        v.begin(),  
        v.end(),  
        [&](auto e) { return f(e1, e); }  
    );  
    cout << cnt;  
}
```

Est $O(n^2)$ )

Votre réponse est correcte.

La réponse correcte est :

```
vector<int> v;  
int cnt = 0;  
for (auto&& e1 : v) {  
    cnt += count_if(  
        v.begin(),  
        v.end(),  
        [&](auto e) { return f(e1, e); }  
    );  
    cout << cnt;  
}
```

Est $O([n^2])$



Question **4**

Correct

Note de 2,00
sur 2,00

Cette question devrait être faite sans utiliser un compilateur.

On désire déterminer l'ordre de construction des différents objets/sous-objets pour une déclaration donnée. On ne listera pas dans cet ordre la "construction" des types fondamentaux, qui n'ont pas de constructeur.

Soit les définitions de classes suivantes (dans des fichiers séparés et on suppose que les "include" sont faits correctement pour que ça compile):

```
class A {  
public:  
    A() {}  
};
```

```
class B {  
public:  
    B() {}  
};
```

```
class C {  
public:  
    C() {}  
};
```

```
class D : public B, public A {  
public:  
    D() { att1_ = new C(); }  
private:  
    C* att1_;  
    A att2_;  
};
```

Soit le programme suivant:

```
int main() {  
    D x;  
}
```

On veut savoir l'ordre de construction lorsqu'on exécute ce programme. Indice: il y a 5 objets/sous-objets construits.



Indiquez uniquement les noms des types dans le bon ordre, exemple: A B C D A

Réponse : (régime de pénalités : 0 %)

Indiquez l'ordre de début de construction pour les classes ci-haut:

D B A A C

Indiquez l'ordre de début d'exécution du corps des constructeurs pour les classes ci-haut:

B A A D C

Réponse bien enregistrée: DBAAC, BAADC

Ce message vert n'indique pas si la réponse est bonne ou non, seulement que toutes les cases ont été remplies:

Tous les tests ont été réussis ! ✓

Correct

Note pour cet envoi : 2,00/2,00.



Vous décidez d'implémenter un planificateur de tâches en utilisant les nouvelles connaissances que vous avez acquises sur la STL.

Les définitions de classes suivantes (se trouvant dans leur .h respectif) représentent la base de votre planificateur, mais nous allons en implémenter seulement une petite partie:

```
class Tache {
public:
    Tache(enums::Priorite priorite, const std::string& nom, const std::tm& echeance = tm());

    bool operator==(const Tache& tache);

    bool operator<(const enums::Priorite& priorite);
    friend bool operator<(const enums::Priorite& priorite, const Tache& tache);
    bool operator<(const tm& echeance);
    friend bool operator<(const tm& echeance, const Tache& tache);

    std::string getNom() const;
    tm getEcheance() const;
    enums::Priorite getPriorite() const;

    friend std::ostream& operator<<(std::ostream& out, const Tache& tache);

private:
    std::string nom_;
    enums::Priorite priorite_;
    tm echeance_;
};
```

```

class Projet {
public:
    Projet(std::string nom);

    std::string getNom() const;

    void ajouterTache(const Tache& tache);
    void supprimerTache(const Tache& tache);

    void trierTaches(std::function<bool(const Tache&, const Tache&> f);
    std::pair<std::vector<Tache>::iterator, std::vector<Tache>::iterator> getTaches(tm& echeance);
    std::pair<std::vector<Tache>::iterator, std::vector<Tache>::iterator> getTaches(enums::Priorite
prio);

    friend std::ostream& operator<<(std::ostream& out, const Projet& projet);

private:
    std::string nom_;
    std::vector<Tache> taches_;

    template<typename Comparateur, typename T>
    std::pair<std::vector<Tache>::iterator, std::vector<Tache>::iterator> getRange(T element);
};

```

```

class Utilisateur {
public:
    void ajouterProjet(const std::shared_ptr<Projet>& projet);

    template<typename Comparateur, typename T>
    std::multimap<Tache, std::string, Comparateur> getTaches(T element);

    std::multimap<Tache, std::string, ComparateurEcheance> getTachesParPriorite(enums::Priorite prio);
    std::multimap<Tache, std::string, ComparateurPrioriteDecroissant> getTachesParEcheance(tm& echeance);

private:
    std::vector<std::shared_ptr<Projet>> projets_;
};

```

Les priorités sont définies sous forme d'énumération:

```

namespace enums {
    enum Priorite {
        BASSE,
        MOYENNE,
        HAUTE,
    };
}

```



ComparateurEcheance, ComparateurPrioriteCroissant et ComparateurPrioriteDecroissant sont des foncteurs qui permettent la comparaison entre deux tâches de la manière énoncée dans leur nom.

Description

Dans les questions de programmation suivantes, vous devez écrire les méthodes dans le .cpp correspondant (hors de la déclaration de la classe).

Pour éviter les redondances, faites appel le plus souvent possible aux méthodes définies dans les classes.

Toutes les méthodes implémentées devront faire l'utilisation des algorithmes de la STL. Aucune boucle (ex. **for**, **while**, **do while**) n'est permise, sauf si l'utilisation de boucles est explicitement permise dans l'énoncé de la question.

Vous n'avez pas besoin d'ajouter le `std::`, l'instruction `using namespace std` est présente dans le code.

Vous avez autant d'essais que nécessaire pour faire compiler le code. Pour la vérification des tests, un régime de pénalité peut s'appliquer, veuillez vérifier l'information pour chaque question de cet exercice.



Question **5**

Incorrect

Note de 0,00
sur 1,50

On vous demande de faire l'implémentation des méthodes ajouterTache et supprimerTache de la classe **Projet**. La tâche est ajoutée au vecteur seulement si elle n'existe pas déjà dans celui-ci.

Par exemple:

Test	Résultat
<pre>projet.ajouterTache(tache1); projet.ajouterTache(tache1); cout << projet.getNbTaches() << endl;</pre>	1
<pre>projet.ajouterTache(tache1); projet.ajouterTache(tache2); cout << projet.getNbTaches() << endl;</pre>	2
<pre>projet.ajouterTache(tache1); projet.ajouterTache(tache2); projet.supprimerTache(tache1); projet.supprimerTache(tache2); cout << projet.getNbTaches() << endl;</pre>	0

Réponse : (régime de pénalités : 10, 20, ... %)

Réinitialiser la réponse

```
11 ▼ {
12     auto it = remove(taches_.begin(), taches_.end(), tache);
13     taches_.erase(it, taches_.end());
14 }
15 ▼ /*The line auto it = remove(taches_.begin(), taches_.end())
16 algorithm from the <algorithm> header to remove all occurrences
17 The remove() algorithm takes three arguments: the beginning
18 the range to operate on, and the value to remove. It returns
19 In the code snippet you provided, the remove() algorithm returns
20 the element past the last element not removed, which is the end of the
21 The auto keyword allows the type of it to be automatically deduced
22 In this case, it will be of type std::vector<Tache>::iterator
23 The it iterator is then used as the starting point to erase elements
24 member function of the taches_ vector. The second argument is the
25 This removes all the elements from the it iterator to the end of the
26 */
27 // Ou, puisque ce n'est pas dit s'il faut enlever plusieurs tâches
28 ▼ void Projet::supprimerTache(const Tache& tache) {
```



```

29     if (auto it = find(taches_.begin(), taches_.end(), tache); it != taches_.end())
30         taches_.erase(it);
31 }

```

Erreur(s) de syntaxe

__tester__.cpp:112:6: error: redefinition of 'void Projet::supprimerTache(const Tache&)'

```
void Projet::supprimerTache(const Tache& tache) {
```

~~~~~

\_\_tester\_\_.cpp:94:6: note: 'void Projet::supprimerTache(const Tache&)' previously defined here

```
void Projet::supprimerTache(const Tache& tache)
```

~~~~~

Solution de l'auteur de la question (Cpp):

```

1 void Projet::ajouterTache(const Tache& tache) {
2     auto it = find(taches_.begin(), taches_.end(), tache);
3     if (it != taches_.end()) return;
4
5     taches_.push_back(tache);
6 }
7
8 void Projet::supprimerTache(const Tache& tache) {
9     auto it = remove(taches_.begin(), taches_.end(), tache);
10    taches_.erase(it, taches_.end());
11 }

```

Incorrect

Note pour cet envoi : 0,00/1,50.



Question **6**

Correct

Note de 1,00
sur 1,00

On vous demande de faire l'implémentation de la méthode trierTaches de la classe **Projet**. La méthode trierTaches permet de trier les tâches du projet en utilisant la fonction qui lui est passée en paramètre.

Par exemple:

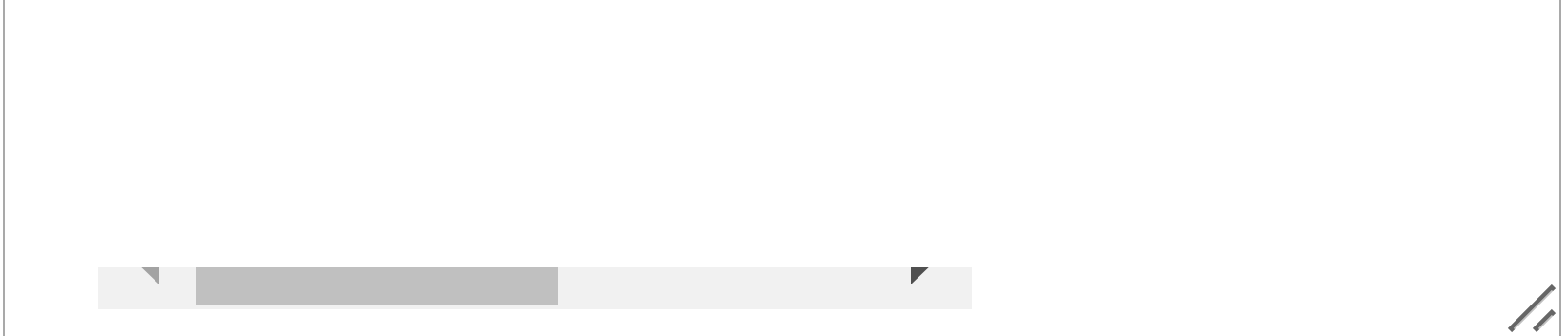
Test	Résultat
<pre>projet.trierTaches(CompareurEcheance()); cout << projet << endl;</pre>	<pre>Nom: INF1010 Taches: Nom: Composition & agregation Priorite: 0 Echeance: 2020/02/12 Nom: Revision Intra Priorite: 2 Echeance: 2020/03/14 Nom: Pointeurs intelligents Priorite: 1 Echeance: 2020/04/20 Nom: Nouvelle tache Priorite: 2 Echeance: 2020/04/20</pre>

Réponse : (régime de pénalités : 10, 20, ... %)

Réinitialiser la réponse

```
1 void Projet::trierTaches(function<bool(const Tache&, const Ta  
2 {  
3     // Attention que le compilateur sur Moodle n'a pas les ra  
4     sort(taches_.begin(), taches_.end(), f);  
5 }  
6 /*n this implementation, the sort() function is called with t  
7 along with the provided comparison function f.  
8 The comparison function f takes two Tache objects as input an  
9 The function returns true if the first argument is less than  
10 and false otherwise. The sorting order is determined by the i  
11 */  
12
```





	Test	Résultat attendu	Résultat obtenu	
✓	<pre>projet.trierTaches(CompareteurPrioriteCroissant()); cout << projet << endl;</pre>	<p>Nom: INF1010 Taches: Nom: Composition & agregation Priorite: 0 Echeance: 2020/02/12</p> <p>Nom: Pointeurs intelligents Priorite: 1 Echeance: 2020/04/20</p> <p>Nom: Revision Intra Priorite: 2 Echeance: 2020/03/14</p> <p>Nom: Nouvelle tache Priorite: 2 Echeance: 2020/04/20</p>	<p>Nom: INF1010 Taches: Nom: Composition & agregation Priorite: 0 Echeance: 2020/02/12</p> <p>Nom: Pointeurs intelligents Priorite: 1 Echeance: 2020/04/20</p> <p>Nom: Revision Intra Priorite: 2 Echeance: 2020/03/14</p> <p>Nom: Nouvelle tache Priorite: 2 Echeance: 2020/04/20</p>	✓



	Test	Résultat attendu	Résultat obtenu	
✓	<pre>projet.trierTaches(CompareteurEcheance()); cout << projet << endl;</pre>	<pre>Nom: INF1010 Taches: Nom: Composition & agregation Priorite: 0 Echeance: 2020/02/12 Nom: Revision Intra Priorite: 2 Echeance: 2020/03/14 Nom: Pointeurs intelligents Priorite: 1 Echeance: 2020/04/20 Nom: Nouvelle tache Priorite: 2 Echeance: 2020/04/20</pre>	<pre>Nom: INF1010 Taches: Nom: Composition & agregation Priorite: 0 Echeance: 2020/02/12 Nom: Revision Intra Priorite: 2 Echeance: 2020/03/14 Nom: Pointeurs intelligents Priorite: 1 Echeance: 2020/04/20 Nom: Nouvelle tache Priorite: 2 Echeance: 2020/04/20</pre>	✓

Tous les tests ont été réussis ! ✓

Solution de l'auteur de la question (Cpp):

```

1 void Projet::trierTaches(function<bool(const Tache&, const
2     sort(taches_.begin(), taches_.end(), f);
3 }
4
```

Correct

Note pour cet envoi : 1,00/1,00.



Soit le code suivant:

```
class Cmp {
public:
    template<typename T>
    bool operator()(const T& gauche, const T& droite) const {
        return gauche > droite;
    }
};

template<typename Ta, typename Tb>
class Noeud {
public:
    Noeud(Ta a, Tb b) : a_(a), b_(b) {}

    bool operator<(const Noeud& n) const {
        return (a_ < n.a_) || (a_ == n.a_ && b_ < n.b_);
    }

    bool operator>(const Noeud& n) const {
        return (a_ > n.a_) || (a_ == n.a_ && b_ > n.b_);
    }

private:
    Ta a_;
    Tb b_;
};

int main() {
    // Le numéro en commentaire à la fin de chaque ligne indique
    // l'ordre d'entrée des éléments dans la map
    map<Noeud<string, string>, int> johiver = {
        {"Etats-Unis", "Lake Placid"}, 1980, // 1
        {"Yougoslavie", "Sarajevo"}, 1984, // 2
        {"Canada", "Calgary"}, 1988, // 3
        {"France", "Albertville"}, 1992, // 4
        {"Norvege", "Lillehammer"}, 1994, // 5
        {"Japon", "Nagano"}, 1998, // 6
        {"Etats-Unis", "Salt Lake City"}, 2002, // 7
        {"Italie", "Turin"}, 2006, // 8
        {"Canada", "Vancouver"}, 2010 // 9
    };

    map<Noeud<string, string>, int, Cmp> johiver2;
    copy(johiver.begin(), johiver.end(), inserter(johiver2, johiver2.begin()));

    return 0;
}
```



Question **7**

Correct

Note de 1,00
sur 1,00

Indiquez, en utilisant les numéros (1 à 9) des entrées dans le conteneur **johiver**, dans quel ordre sont stockés les éléments dans **johiver**.

Dans la réponse, indiquez la série de chiffres sans espaces entre eux.

Réponse :



Dans une map, les éléments sont triés (par défaut) par ordre croissant de leur clé. Dans ce code, la clé est de type Noeud<string, string>. Il faut donc vérifier la surcharge de l'opérateur < de la classe Noeud. Celui ci compare d'abord les attributs a_ (de type string dans notre cas): par défaut les string sont comparés selon leur ordre chronologique. Si les attributs a_ sont identiques, ce sont les attributs b_ qui sont comparés.

Ainsi, les entrées dans johiver sont stockées par ordre alphabétique de pays, puis de ville.

La réponse correcte est : 391748652

Question **8**

Correct

Note de 1,00
sur 1,00

Indiquez, en utilisant les numéros (1 à 9) des entrées dans la map **johiver**, dans quel ordre sont stockés les éléments dans **johiver2**.

Dans la réponse, indiquez la série de chiffres sans espaces entre eux.

Réponse :



johiver2 est une copie de johiver à la seule différence que johiver2 est déclarée comme une map dont le prédicat de comparaison n'est pas celui par défaut, mais un foncteur Cmp. Il faut donc vérifier l'ordre de tri de Cmp. Celui-ci utilise l'opérateur > sur les éléments de la map. Dans ce code, la clé est de type Noeud<string, string>. Il faut donc vérifier la surcharge de l'opérateur > de la classe Noeud. Celui ci compare d'abord les attributs a_ (de type string dans notre cas): par défaut les string sont comparés selon leur ordre chronologique. Si les attributs a_ sont identiques, ce sont les attributs b_ qui sont comparés.

Ainsi, dans johiver2, les éléments sont stockés dans leur alphabétique inverse des pays, puis des villes.

La réponse correcte est : 256847193



Question **9**

Correct

Note de 1,00
sur 1,00

Dans le programme principal précédent, quelle version de l'opérateur générique () de la classe Cmp est appelée (en d'autres termes, à quel type correspond le T):

Veuillez choisir une réponse.

- ☐ a. pair<Noeud<string, string>, int>
- ☒ b. Noeud<string, string> ✓
- ☐ c. string
- ☐ d. int

Votre réponse est correcte.

La réponse correcte est : Noeud<string, string>

Question **10**

Correct

Note de 1,00
sur 1,00

Cochez les affirmations qui sont justes.

Veuillez choisir au moins une réponse.

- ☒ a. Il est possible de lever une exception dans un bloc de capture d'exception ("catch"). ✓
- ☒ b. Il est possible d'utiliser plusieurs blocs "catch" à la suite d'un bloc "try". ✓
- ☐ c. Après la capture et la gestion d'une exception, le programme reprend son exécution à l'instruction ayant causé l'exception.
- ☐ d. Lever une exception dans un destructeur sans la gérer permet d'éviter les fuites de mémoire.
- ☐ e. Lors de la levée d'une exception, toute la mémoire dynamiquement allouée par la fonction est automatiquement libérée.

Votre réponse est correcte.

Les réponses correctes sont : Il est possible de lever une exception dans un bloc de capture d'exception ("catch")., Il est possible d'utiliser plusieurs blocs "catch" à la suite d'un bloc "try".



Question **11**

Correct

Note de 0,75
sur 0,75

L'utilisation d'un espace de noms (namespace) permet de réduire ✓ la portée (scope) de ce qu'elle contient.

Il est possible de définir un seul ✓ symbole(s) ayant le même nom dans un même espace de nom.

Votre réponse est correcte.

La réponse correcte est :

L'utilisation d'un espace de noms (namespace) permet de [réduire] la portée (scope) de ce qu'elle contient.

Il est possible de définir [un seul] symbole(s) ayant le même nom dans un même espace de nom.



Question **12**

Partiellement
correct

Note de 0,75
sur 1,00

Cochez les affirmations qui sont justes.

Veuillez choisir au moins une réponse.

- ☒ a. L'utilisation d'attributs statiques de classe permet de les partager entre toutes les instances de la classe. ✓
- ☒ b. L'accès à une méthode statique peut se faire à l'aide de l'opérateur de résolution de portée ("::"). ✓
- ☐ c. Utiliser le mot clef static permet de réduire la portée (scope) d'une méthode.
- ☒ d. Une méthode statique peut être appelée sans instancier d'objet de la classe dont elle fait partie. ✓
- ☐ e. Un attribut statique de classe est forcément public.
- ☐ f. Un attribut non statique de classe peut être directement accédé dans une méthode statique de la même classe, comme pour toute autre méthode de cette classe.
- ☒ g. Une variable globale statique est accessible dans d'autres fichiers que celui dans lequel elle est déclarée. ✗

Votre réponse est partiellement correcte.

Vous avez sélectionné trop d'options.

Les réponses correctes sont : L'utilisation d'attributs statiques de classe permet de les partager entre toutes les instances de la classe., Une méthode statique peut être appelée sans instancier d'objet de la classe dont elle fait partie., L'accès à une méthode statique peut se faire à l'aide de l'opérateur de résolution de portée ("::").



Lors de la définition d'une nouvelle exception par l'utilisateur, celle-ci doit absolument hériter de `std::exception`.

Veuillez choisir une réponse.

- ☐ Vrai
- ☒ Faux ✓

La réponse est "faux", les exceptions peuvent être des classes à part entière ou des POD (int, double etc.). Cependant, il est souvent préférable de les faire hériter de `std::exception`.

Dériver de `std::exception` permet de conserver une interface standard dans la gestion d'exception. De plus lors de la gestion d'exception, l'utilisateur peut simplement capturer les exception de types `std::exception`. En général on ne dérive pas directement de `std::exception` mais plutôt de ses classes filles telles que `std::runtime_error` ou `logic_error` comme vu dans le cours.

La réponse est "faux", les exceptions peuvent être des classes à part entière ou des POD (int, double etc.). Cependant, il est souvent préférable de les faire hériter de `std::exception`.

Dériver de `std::exception` permet de conserver une interface standard dans la gestion d'exception. De plus lors de la gestion d'exception, l'utilisateur peut simplement capturer les exception de types `std::exception`. En général on ne dérive pas directement de `std::exception` mais plutôt de ses classes filles telles que `std::runtime_error` ou `logic_error` comme vu dans le cours.

La réponse correcte est « Faux ».



Soit le code suivant (des commentaires sont présents à la place de certaines lignes de code pour alléger la lecture) :



```

1. class ErreurCovid19 : public runtime_error {
2. public:
3.     using runtime_error::runtime_error;
4. };
5.
6. class ErreurQuarantaine : public ErreurCovid19 {
7. public:
8.     using ErreurCovid19::ErreurCovid19;
9. };
10.
11. class ErreurDistanciationSociale : public ErreurCovid19 {
12. public:
13.     using ErreurCovid19::ErreurCovid19;
14. };
15.
16. class Citoyen {
17. public:
18.     void faireDesTartelettesPortugaises() {
19.         if (not vérifierGardeManger())
20.             faireDesCourses();
21.
22.         // Faire la recette...
23.
24.         cout << "Miam!" << "\n";
25.     }
26.
27.     void faireDesCourses() {
28.         if (estSymptomatique())
29.             throw ErreurQuarantaine("Peut pas faire l'épicerie :(");
30.
31.         bool fini = false;
32.         while (not fini) {
33.             if (mesurerDistancePlusProche() < 2.0)
34.                 throw ErreurDistanciationSociale("Stranger danger!");
35.
36.             // Se promener et acheter ce qu'il faut...
37.             // Mettre à jour 'fini', quand on a ce qu'il faut
38.         }
39.
40.         // Après appel sans erreur de la méthode, vérifierGardeManger() retourne true.
41.     }
42.
43.     bool vérifierGardeManger() const;
44.     double mesurerDistancePlusProche() const;
45.     bool estSymptomatique() const;
46.     void attendre14Jours();

```



```
47.     void    faireAttentionAu2m();
48. };
49.
50.
51. int main() {
52.     Citoyen drArruda;
53.     // Faire de la distanciation sociale...
54.
55.     try {
56.         drArruda.faireDesTartelettesPortugaises();
57.     } catch (ErreurDistanciationSociale&) {
58.         drArruda.faireAttentionAu2m();
59.     } catch (ErreurQuarantaine&) {
60.         drArruda.attendre14Jours();
61.     } catch (ErreurCovid19& e) {
62.         cout << "Erreur de COVID-19 : " << e.what() << endl;
63.     } catch (exception& e) {
64.         cout << "Erreur générique : " << e.what() << endl;
65.     } catch (...) {
66.         cout << "Erreur inconnue" << endl;
67.     }
68. }
69.
```



Quel est le comportement du code ci-dessus si le citoyen n'a pas tous les ingrédients nécessaires (la méthode `vérifierGardeManger()` retourne `false`) et qu'il est malade (si la méthode `estSymptomatique()` retourne `true`)? On suppose que les bouts de code remplacés par des commentaires et les méthodes non implémentées ne lancent pas d'exception.

Veuillez choisir une réponse.

- ☐ a. Une exception est lancée dans `Citoyen::faireDesCourses()`, se propage à travers `Citoyen::faireDesTartelettesPortugaises()` et est attrapée dans le `main()`. Étant donné qu'on a mis un `catch(...)`, c'est celui-ci qui attrape toutes les erreurs. On a donc un message d'erreur inconnue qui s'affiche et le programme se termine correctement (atteint la fin du `main()`).
- ☐ b. Une exception est lancée dans `Citoyen::faireDesCourses()`, se propage à travers `Citoyen::faireDesTartelettesPortugaises()` et est attrapée dans le `main()`. Le message `Erreur de COVID-19` est affiché (troisième `catch`).
- ☐ c. Une exception est lancée dans `Citoyen::faireDesCourses()`, se propage à travers `Citoyen::faireDesTartelettesPortugaises()` et à travers le `main()`. Le programme plante à cause d'une exception non gérée.
- ☒ d. Une exception est lancée dans `Citoyen::faireDesCourses()`, se propage à travers `Citoyen::faireDesTartelettesPortugaises()` et est attrapée dans le `main()`. Le citoyen va ensuite attendre 14 jours (deuxième `catch`) et le programme se termine correctement (atteint la fin du `main()`). 


Votre réponse est correcte.

La réponse correcte est : Une exception est lancée dans `Citoyen::faireDesCourses()`, se propage à travers `Citoyen::faireDesTartelettesPortugaises()` et est attrapée dans le `main()`. Le citoyen va ensuite attendre 14 jours (deuxième `catch`) et le programme se termine correctement (atteint la fin du `main()`).



Quel est le comportement du code ci-dessus si le citoyen n'a pas tous les ingrédients et que durant ses courses il est mal distancé des autres citoyens (`mesurerDistancePlusProche()` retourne une valeur < 2)?

Veuillez choisir une réponse.

- ☒ a. Une exception est lancée dans `Citoyen::faireDesCourses()`, se propage à travers `Citoyen::faireDesTartelettesPortugaises()` et est attrapée dans le `main()`. Le citoyen va ensuite garder ses distances et le programme se termine correctement (atteint la fin du `main()`). 
- ☐ b. Une exception est lancée dans `Citoyen::faireDesCourses()`, se propage à travers `Citoyen::faireDesTartelettesPortugaises()` et est attrapée dans le `main()`. Le citoyen va ensuite attendre 14 jours et le programme se termine correctement (atteint la fin du `main()`).
- ☐ c. Une exception est lancée dans `Citoyen::faireDesCourses()`, se propage à travers `Citoyen::faireDesTartelettesPortugaises()` et est attrapée dans le `main()`.. Étant donné qu'on a mis un `catch(...)`, c'est celui-ci qui attrape toutes les erreurs. On a donc un message d'erreur inconnue qui s'affiche et le programme se termine correctement (atteint la fin du `main()`).
- ☐ d. Une exception est lancée dans `Citoyen::faireDesCourses()`, se propage à travers `Citoyen::faireDesTartelettesPortugaises()` et est attrapée dans le `main()` par le troisième catch. Une erreur COVID-19 est affichée et le programme se termine correctement (atteint la fin du `main()`).

Votre réponse est correcte.

La réponse correcte est : Une exception est lancée dans `Citoyen::faireDesCourses()`, se propage à travers `Citoyen::faireDesTartelettesPortugaises()` et est attrapée dans le `main()`. Le citoyen va ensuite garder ses distances et le programme se termine correctement (atteint la fin du `main()`).



Question 16

Correct

Note de 1,50
sur 1,50

On considère l'implémentation suivante du `main()` en conservant le reste du code inchangé :

```
1. int main() {
2.     Citoyen drArruda;
3.     // Faire de la distanciation sociale...
4.
5.     while (true) {
6.         try {
7.             drArruda.faireDesTartelettesPortugaises();
8.             break;
9.         } catch (ErreurDistanciationSociale&) {
10.            drArruda.faireAttentionAu2m();
11.        } catch (ErreurQuarantaine&) {
12.            drArruda.attendre14Jours();
13.        } catch (ErreurCovid19& e) {
14.            cout << "Erreur de COVID-19 : " << e.what() << endl;
15.        } catch (exception& e) {
16.            cout << "Erreur générique : " << e.what() << endl;
17.        } catch (...) {
18.            cout << "Erreur inconnue" << endl;
19.        }
20.    }
21. }
```

Supposons que la méthode `estSymptomatique()` retourne vrai une première fois, puis retourne faux par la suite, et que la méthode `vérifierGardeManger()` retourne faux au départ, puis retourne vrai après un appel complet à `faireDesCourses()`. Qu'arrive-t-il?

Veuillez choisir une réponse.

- ☐ a. Au premier appel de `drArruda.faireDesTartelettesPortugaises()`, une exception est levée et attrapée par le deuxième `catch`. Le `try-catch` brise tout de suite la boucle et le programme se termine normalement.
- ☒ b. Au premier appel de `drArruda.faireDesTartelettesPortugaises()`, une exception est levée et attrapée par le deuxième `catch` (donc on attend 14 jours). La boucle se répète une deuxième fois, la méthode complète son appel et affiche «Miam!», puis la boucle se brise. Le programme se termine ensuite normalement. ✓
- ☐ c. Au premier appel de `drArruda.faireDesTartelettesPortugaises()`, une exception est levée et attrapée par le `catch(...)` qui attrape toutes les erreurs. Toutefois, puisque le `try-catch` est dans un `while`, on obtiendra nécessairement une boucle infinie dans la majorité des cas.



Votre réponse est correcte.

La réponse correcte est : Au premier appel de `drArruda.faireDesTartelettesPortugaises()`, une exception est levée et attrapée par le deuxième `catch` (donc on attend 14 jours). La boucle se répète une deuxième fois, la méthode complète son appel et affiche «Miam!», puis la boucle se brise. Le programme se termine ensuite normalement.

◀ Réponses aux questions de pratique

Aller à...

Réponses aux questions de pratique
pour final ▶

